Steganography Analysis: Efficacy and Response-Time of Current Steganalysis Software

Jordan Green[1], B.S., Ian Levstein[1], M.S., Cpl. Robert J. Boggs[2], Terry Fenger[1], Ph.D.
[1]Marshall University Forensic Science Center
[2]West Virginia State Police Digital Forensics Unit

Abstract: Steganography, Latin for "covered writing," is a method of hiding information within digital media. It is a method of hiding information in plain sight without detection from unintended recipients. In steganography, a message is embedded into a carrier or host file through means such as least significant bit encoding, appending, or watermarking. Many file types including audio, video, image, and text can be embedded into carrier files of equally diverse formats. Applications, which are downloadable from the Internet, easily create steganography, and simple passwords unlock the messages within. Today, steganography grows more complex with an increase in such open-source applications, which hide data. As applications become more sophisticated the need to detect, analyze, and stop the flow of dangerous information becomes more crucial. Due to the increasing need for steganalysis software, companies like BackBone Security have developed programs that detect and decode steganography. StegAlyzer™ is a software program that detects and analyzes suspect files in order to aid law enforcement in the discovery of evidence that may condemn criminals. While there are four programs within the StegAlyzer™ suite, this investigation dealt with its Signature Search (StegAlyzerSS™) and Artifact Scanner (StegAlyzerSS™) due to their abilities to detect steganography applications and the steganography created from these applications. Several questions were asked in this study: does analysis time change with different carrier and message sizes and formats, how well does StegAlyzerAS™ detect multiple steganography applications, and can StegAlyzerSS™ detect steganography from these applications? For the first question, a free application named GhostHost was selected to create steganography of differing sizes and formats. The open-source steganography applications chosen for the latter two questions were GhostHost, ImageSpyer G2, OpenStego, Steg, Steganography Studio, Open Puff, Silent Eye, Steghide, and Secret Layer. StegAlyzerAS™ was able to identify signatures from five out of nine applications investigated in this study and StegAlyzerSS had a success rate of 33% in identifying steganography created by the applications. StegAlyzerSS™ was also used to analyze the duration of detection for image steganography created by GhostHost, a steganography appending, open-source application. Analysis-time fell within the range of 0.15 and 0.25 second regardless of carrier or message file size. A one-way analysis of variance showed that different carrier and message sizes and formats had no statistical effect on analysis-time. Further studies should investigate StegAlyzer™'s abilities compared to other steganalysis software, such as WetStone's StegoHunt™ or open-source steganalysis software such as Steganography Studio. StegAlyzer™ is an invaluable tool for investigations of digital crimes, and requires competent analysts to be effective.

Key Words: Steganography, Steganalysis, Investigation

Introduction

Hidden communication has a long history, and as our verbal and written communication has grown more complex, so has our ability to keep information hidden. Cryptography and steganography are the products of hidden communication using text or images. While cryptography is any encoded message, steganography is often more complex. Steganography stems from the Greek root: “stegos,” or cover, and “grafia,” or writing [4]; it literally translates to “covered writing” [11]. Steganography’s use is directly related to its translation: hiding messages within other messages.

The purpose of steganography is to conceal and prevent detection of a secondary, often unrelated message within an innocent picture or text. Steganography is used by different groups, and thus may be vehicles for personal privacy or illicit proliferation of data [2]. Cryptography encodes visible information like bank statements into an unreadable format in an effort to prevent fraud or tampering; steganography also encodes such information, but additionally hides the presence of the sensitive data [1]. Due to a perceived lack of security by many Internet users, steganography has become a viable option to protect sensitive information; on the other hand, steganography also makes it possible to hide illegal information as well. The issue of detection from a law-enforcement perspective leads to many problems with the spread of steganography [2]. For example, it is believed that Bin Laden and recent terrorist cells used steganography within images to disperse maps and targets [18]. So, while steganography can be a useful tool to hide important personal information, in the wrong environment it can also be dangerous.

Steganography was developed after the success of cryptography, or coded writing [18]. Cryptography encodes a message by re-ordering the letters into an unreadable format, while steganography embeds a message within another file, such as an image or audio recording. An early example of cryptography is Caesar's letter substitution in government writings. Later, the Greeks used steganography by scribing messages on slaves' shaved heads, allowing the hair to grow, and then sending the slaves to the recipients of the messages. The most secure way of hiding information is a combination of steganography and cryptography or even more complex innovations like quantum cryptography, which combines cryptography and physics to alert senders and recipients of intruders [11]. While these more advanced methods of cryptic messaging are useful today, modern steganography got its start in the 1980's.

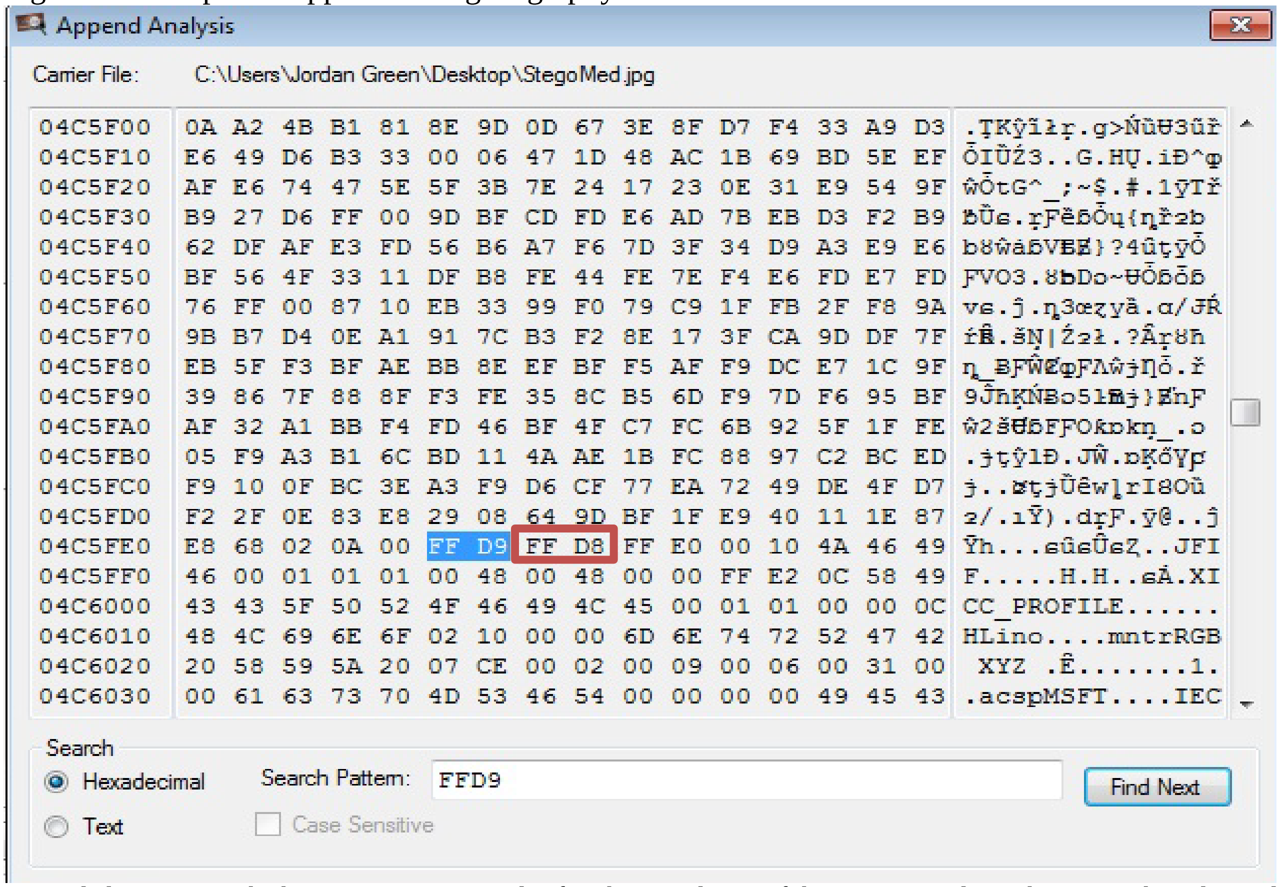
Steganography became more complex with the advent of computers, and in 1985 it was implemented through methods such as invisible ink, embedded pictures in video material, and concealed data via encryption. By the mid 90's steganography evolved into today's most common forms: pure steganography, secret key steganography, and public key steganography [2]. Pure steganography is the least secure method, because software decryption is readily available. Secret key steganography is more secure because a key, which decodes the algorithms used to hide the information, is shared between the sender and recipient. If the key is kept secure, only those two parties can decipher the message. Finally, public key steganography uses a public key to encode the information, which is shared between the receiver and the intended recipients of the steganography. In order to extract the message within the steganography, a private key must be used. That is, the public key imbeds the information within the carrier file, and the private key extracts said information. Security is increased because two separate keys are used, which naturally decreases the chances of both being compromised to unintended recipients.

Today, most steganography creators use the public key method, and choose from over 1,500 steganography applications available on the Internet [2].

The two main components of steganography are the message and the carrier of the message. Intuitively, the message is what is intended for the receiver of the steganography. Messages are usually text or images, and in most cases they must be smaller in size than the carrier. An exception to this rule is appended steganography, which does not have a size requirement for the message file. The size of the message is referred to as its payload, which determines the size requirement of the image in which it is embedded. Generally, payloads must be no more than 16% the size of the carrier, or detection of the steganography may occur. The medium in which the hidden message is embedded is termed a carrier [11]. Carriers have also been referred to as cover media or the host media/signal. The carrier is what the general public sees when the file is opened. Together, the carrier and message comprise the steganography, or stego [18] and to reveal the message a unique key must be implemented [11]. Carriers can be text, image, audio, or video files and keys are included within the algorithm that created the steganography. Recently, steganography has been embedded within packets on a network, though further research must be done to confirm this [18]. Usually, text files are poor carriers due to ease of alteration and their compact size. Carrier files are often images because the changes that occur while steganography is being embedded are generally undetected by the human eye, can be variable in size, and are not often manipulated [2].

The method for embedding a message within an image carrier is most commonly one of three methods: altering the least significant bit (LSB), masking and filtering techniques (which includes watermarking) [2], or appending steganography code to the end of an image file's code [6]. Altering the least significant bit places the message's bits into the least important bit of a byte. That is, a one or a zero that is non-native to the carrier is placed at the end of a string of ones and zeroes that are; this is how the message is built. Therefore, for every eight bits (or byte) within the carrier, the least important bit is part of the hidden message. Because the bits of a pixel are being changed (for an image carrier), certain pixels within the image will be altered. Generally, this bit contributes to the brightness of the image or noise within it [18], which may result in a blue pixel being a darker blue, or a green pixel being a lighter green. These changes, while indistinguishable in a very large image, can be detected in a small, 8-bit image. For this reason, it is important to consider the size and complexity of an image when choosing a carrier for a message. Masking and filtering deal mostly with placing code on top of an image to increase its size for accommodation of steganography, but not changing the code of the image. What results is an image that looks tinted or shaded but maintains the integrity of the message upon alteration of the carrier file [2]. Finally, appending code to the end of an image file involves the code being physically attached after the completion of the image's innate code [6]. For instance, a JPEG viewed in a hex-editor ends when the hex value "FF D9" appears. A computer interprets the hex code "FF D9" as the end of the file, so anything that appears beyond "FF D9" has been deliberately appended. For a JPEG image, the steganography code is visible after the "FF D9" hexadecimal string. This makes appended steganography vulnerable in that a hex-editor can easily discern the stego without much in-depth analysis. Figure 1 is an example of appended code after a JPEG image.

Figure 1: Example of Appended Steganography



Appended steganography begins its message code after the completion of the carrier's code, in this case at hexadecimal FF D9. A hex-editor can easily detect appended steganography.

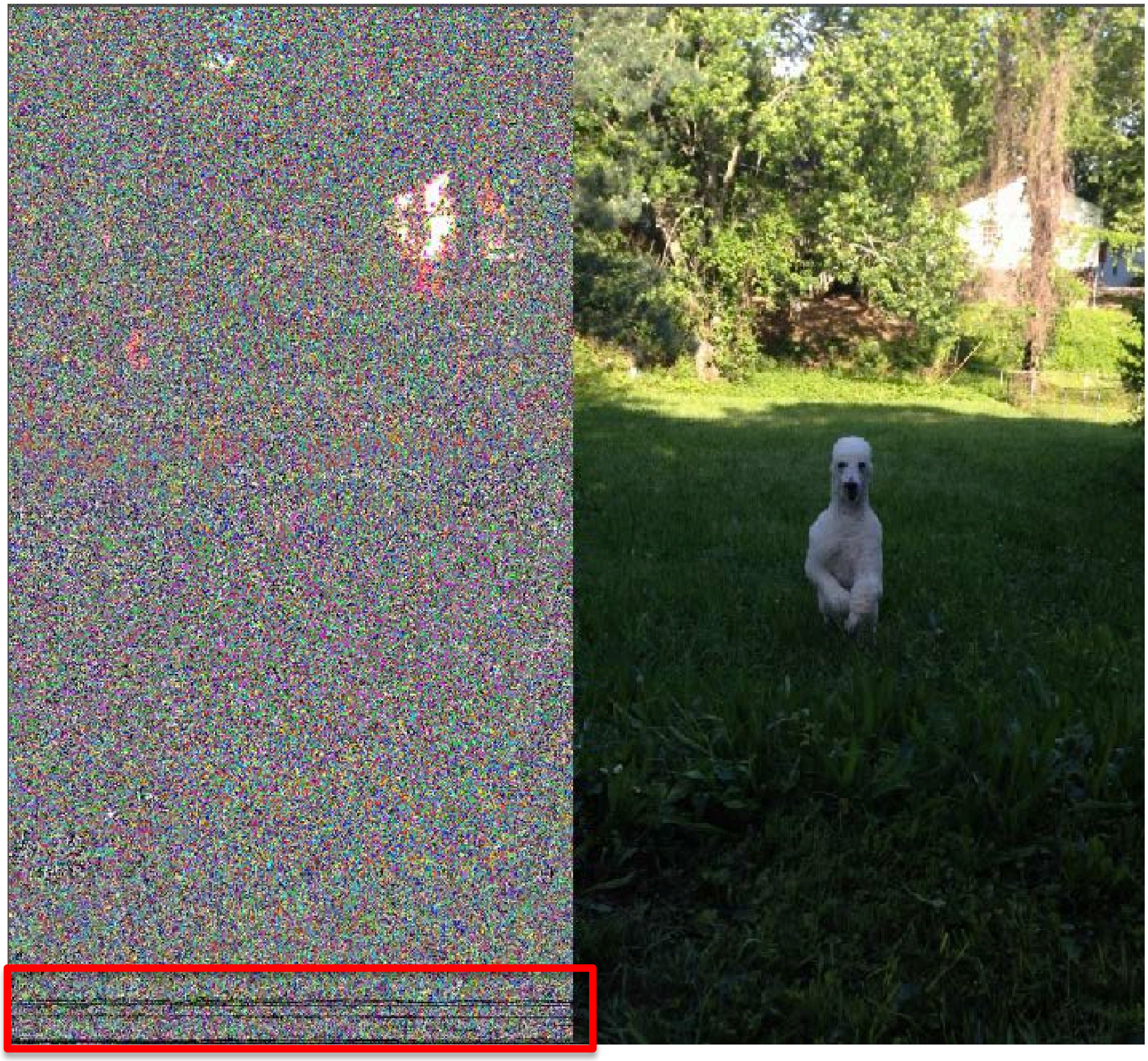
While these techniques are the most common methods for steganography creation in images, audio has also been used as a carrier.

As with image steganography, audio steganography is accomplished through the execution of several techniques, which differ according to when they are applied to the audio file (before or after compression to MP3 format). Three methods are low-bit encoding, phase coding, and spread spectrum coding [3]. Low-bit encoding is much like LSB for images but codes bits according to lapsed time instead of pixels; noise is likely discernable with low-bit encoding [2]; phase-coding embeds information within sound waves and alters the wave in such a way that the noise created is imperceptible [3]; and spread spectrum coding uses the entire frequency spectrum to code and emit the audio [2]. While, audio has a high degree of redundancy within its data and may be transmitted quickly, steganalysis programs are fairly new for audio files. This makes the use of steganography through audio file carriers successful. However, while the immaturity of steganalysis for audio files is an advantage, the immaturity of steganography algorithms that create audio file steganography is a disadvantage. No matter the arrangement or formatting of cover objects and their embedded messages, it is important that all steganography adhere to three basic principles to be effective: imperceptibility, capacity, and robustness [3].

These three fundamental aspects of steganography render it capable or ineffectual vehicles of elusive message transfer. Imperceptibility is the ability of the steganography to go undetected in its carrier object, which is possible through use of unique images, very large images, or audio files with inherent visible or shadow noise. Capacity is the payload, or size of the message being sent. Boora and Gambhir [4] aptly describe capacity and imperceptibility as at odds with each other because as capacity increases imperceptibility decreases and visa versa. Imperceptibility must be minimized and capacity must be maximized in order to optimize the effect of steganography. Finally, robustness is the ease with which a steganography tool may be repeatedly used [4]. For steganography to be successful these three principles must be optimally taken advantage of; otherwise, it may be vulnerable to attack.

The principles of obfuscation are important in that stego, if detected, may be intercepted and decrypted or tampered with and rendered ineffective. Indeed, even ignorant tampering of cover images may render the embedded steganography useless. When tampering is intentional, the process is called steganalysis. There are several means of steganalysis: detection, destruction, extraction, and modification [4]. Detection is subdivided into passive and active. Passive detection does not seek to discover the hidden message within steganography; passive detectors destroy or modify files that are believed to be stego. Active detection is the act of seeking out and manipulating steganography in an effort to uncover the secret within the stego [18]. Message extraction occurs when the algorithm used to create the stego is cracked and used to decipher the message; it is also possible to manually extract data from the image file if the bits used to create the message are known. These detection and modification methods are generally referred to as steganalysis. Modification of stego generally refers to changes made to the carrier file that destroy or cripple the hidden message. Such modifications can be text or formatting changes in a text carrier, deletion of sound bytes from an audio carrier, or alterations such as cropping in an image carrier [4]. Figure 2 illustrates a representation of hidden steganography within an image file. Notice the lattice at the bottom of the image on the left.

Figure 2: Example of Lattice Resulting from LSB Steganography



In least significant bit steganography, a lattice may be observable when running steganalysis software. In this image, the steganography is contained within the bottom portion of the carrier.

From here, the analyst may passively detect the image by destroying or modifying the lattice without extracting the message. Conversely, the analyst may use active detection by using a steganalysis software program to parse the hidden data.

Steganalysis is the detection, and in some cases, the decryption of steganography. There are many steganography tools available to the public, and many of them can go completely undetected by today's steganalysis programs [5]. Generally, steganalysis deals most often with the decryption of stego in order to recover the hidden message within it. These decryption methods take advantage of algorithms within the stego, and use statistical approaches to attack it. Some of these statistical analyses include tests such as the chi-squared test and dual statistical steganalysis. The chi-squared test makes predictions about patterned pixels within an image; if one part of the image has the same intensity level throughout, it is probable that there is steganography within the carrier. While chi-squared testing is relatively basic and can be beaten by randomly assorted LSB stego, a more sophisticated algorithm known as dual statistical steganalysis can predict the quantity of pixels that have been altered or flipped. All of the quantitative attacks for stego take advantage of the fact that changing a bit within a carrier file leaves a statistical trace. Steganalysis software like StegAlyzer™ uses statistical algorithms to attack steganography, but they have weaknesses in their programming due to the fact that stego applications employ varying types of embedding techniques. This problem is rampant with steganalysis, and the "Artificial Neural Network Technology for Steganography" (ANNTS), a university-driven initiative, is the first attempt at creating a catchall steganalysis program [5]. As steganalysis software evolves, evaluation of performance of these techniques lends much to the practicality of their use within law enforcement settings.

Backbone Security developed StegAlyzer™, a premier software package that analyzes signatures of steganography applications and extracts messages from steganography. The company created three different applications: StegAlyzerAS™, StegAlyzerSS™, and StegAlyzerRTS™. The current research used StegAlyzerSS™ and StegAlyzerAS™ to identify steganography and its associated applications on a host hard-drive.

There is a prevalence of open-source, or free, applications on the Internet that allow obfuscation of data, and according to the StegAlyzer™ Web site, there are over 1,500 applications available today [15]. For this reason, the current research analyzed multiple applications to discern the efficacy of StegAlyzer™. The applications used to create the steganography are: Image SpyerG2, OpenPuff, OpenStego, SecretLayer, SilentEye, Steg, GhostHost, Steganography Studio, and Steghide [8-10, 12-17]. These were all downloaded either from their associated Web sites, third-party Web sites, or as an example case through the StegAlyzer™ 30-day free trial.

Research Questions:

It is not uncommon for creators of steganography to embed differently formatted stego within cover files. Image files are the most commonly used covers [5], but it is possible to embed image or text within them. For this reason, the current research investigated stego within JPEG and PNG files containing text and image files. This investigation focused on the duration of analysis for stego files of increasing payload and complexity. In a practical setting, it is important to understand the time required to process a case. This study also investigated StegAlyzerAS™'s ability to find artifacts from different applications. According to Backbone Security's website, StegAlyzerAS™ can identify 1,225 artifacts out of the 1,500 available. Finally, the investigators examined StegAlyzerSS™‘s ability to discover steganography from the applications used in the second analysis. Again, Backbone Security's website reports that StegAlyzerSS™ identifies byte patterns from more than 55 applications. Question 1: does the use of StegAlyzer™ create a backlog when analyzing steganography? The study also investigated StegAlyzer™'s efficacy against various stego applications and their steganography. Question 2: how diverse is StegAlyzerAS™'s library in a contemporary, real-world setting; Question 3: how well does StegAlyzerSS™ respond to multiple sources of steganography? The researchers hypothesize that a change in carrier and message sizes and formats will affect analysis time, that StegAlyzerAS^{TM} will identify artifacts from 80% of the applications installed onto a computer, and that StegAlyzerSS^{TM} will identify more than 3% of the steganography created from applications.

Materials and Methods

Scenario 1: Detection-time of Steganography within Different Media using StegAlyzerSS^{TM}

All applications used in Scenario 1 were downloaded and implemented on a Dell Optiplex 990 with a Core i7-2600 processor, running 64-bit Windows 7 Enterprise operating system. Text documents were created using Microsoft Word 2010. All carrier and message sizes and formats used for the experiment are represented in Appendix A.

Ten different images were duplicated and altered to create the seven batches used to test the analysis-time of StegAlyzer^{TM} for different message formats and sizes. The JPG images were all 5MB in size and each batch contained differently sized and formatted message files. The messages were document files (RouxRunSm.doc, RouxRunLge.doc), Joint Photographic Experts Group (JPG) images (StegoSml.jpg, StegoLge.jpg), and Portable Network Graphics (PNG) images (StegoSml.png and StegoLge.png).The batches were organized as outlined in Appendix B: the Control batch contained no embedded steganography, Batch 1 a 34KB text file (.doc), Batch 2 a 103KB text file (.doc), Batch 3 a 1MB JPEG image, Batch 4 a 10MB JPEG image, Batch 5 a 1MB PNG image, and Batch 6 a 10MB PNG image.

To test analysis-time of different carrier sizes and formats, the same ten images were used, but altered to three different size categories and two formats: 1MB JPG, 5MB JPG, 10MB JPG, 1MB PNG, 5MB PNG, and 10MB PNG. These two image extensions were used because PNG is an uncompressed format, while JPG is a form of lossy compression. These compression differences are important because they impart different size formatting, and thus can affect analysis time for StegAlyzerSS^{TM} software. Further, PNG and JPG are equally ubiquitous image formats, so sampling from them is intrinsically significant to real-world applications. The embedded message file was a 5MB JPG image. The results of this analysis are depicted in Appendix C.

Ghost Host v1.0.1.1 (©1998 Kelce Wilson) was used to create the steganography for each image. Note that GhostHost is a steganography appending application, and not a least significant bit encoding application, so there were no size requirements for the message or carrier files.

All images were captured with an iPhone 5 using iOS v7.1.1. The images were resized using Apple's "Preview" application on a Mac Powerbook running iOS 10 Mavericks. The images were transferred (via a Kingston 16 GB thumb drive) and saved onto the PC in PNG and JPG formats. Original image sizes ranged from 0.98 MB to 9.81 MB, as depicted in Appendix A.

The steganography files were analyzed using Backbone Security's StegAlyzerSS™ v3.91 (x86). Analysis times of each image and batch of images were recorded using the iPhone 5's native stopwatch application by simultaneously activating the stopwatch and StegAlyzerSS™ and deactivating the stopwatch after the appearance of StegAlyzer™'s completion prompt.

Each analysis attempted signature, append, and LSB analyses. JPEG images yielded results for the signature search and append analysis, while PNG images yielded only signature search results. Note: LSB analysis was not expected to occur due to the nature of GhostHost's steganography-appending functionality.

The elapsed times of each batch were then statistically analyzed. Variances of each batch were recorded, and a one-way Analysis of Variance (ANOVA, $\alpha = 0.05$) was performed to discern any differences among the batches (Method 1: N = 70, n = 10; Method 2: N = 120, n =

10). All statistical analyses were conducted on the Optiplex computer using Microsoft Excel 2010.

Scenario 2: Detection of Multiple Steganography Applications using StegAlyzerAS™

All applications used in Scenario 2 were downloaded and implemented on a Dell Inspiron 1520 with an Intel Core™2 Duo processor, running Microsoft Windows 7 Enterprise.

Nine open-source steganography applications were downloaded from the Internet, chosen based their abilities to create image carriers, the amount of inherent malware within the executable files, and the availability of the software from online sources. These applications were: Image SpyerG2, OpenPuff, OpenStego, SecretLayer, SilentEye, Steg, GhostHost, Steganography Studio, and Steghide [8-10, 12-17]. Each of these applications embedded steganography using either the LSB or appending method, and each one created image steganography. These applications were chosen based on the likelihood that most steganography users would find them the easiest, most ubiquitous, and least harmful applications for their systems.

These applications were then added to an empty folder titled "Steganography Applications." The entire folder was subsequently scanned using StegAlyzerAS™.

Scenario 3: Detection of Steganography of varying Applications using StegAlyzerSS™

Steganography was created on the Optiplex 990 from Scenario 1. Six applications were used to create steganography. These applications were: Image SpyerG2, Open Puff, Open Stego Secret Layer, Steg, Ghost Host, and Steganography Studio [8-10, 12, 14, 15, 16]. The same image file (StegoLge.jpg, 9.764 MB) was imbedded with either a .doc file (RouxRunLg.doc, 103 KB) or a .txt file (Roux.txt, 1KB) depending on the capacity of the carrier file assigned by the application. These images were then placed in a folder and transferred to the Inspiron 1520 for analysis with StegAlyzerSS™. From there, StegAlyzerSS™ detected signatures specific to the application used to create the steganography, the use of LSB-steganography, or appended steganography.

Results and Discussion

Scenario 1:

Figure 3 shows each image used in the detection runs and steganography creation. Figure 4 provides the text and images used as stego messages.

Figure 3: Carrier Images

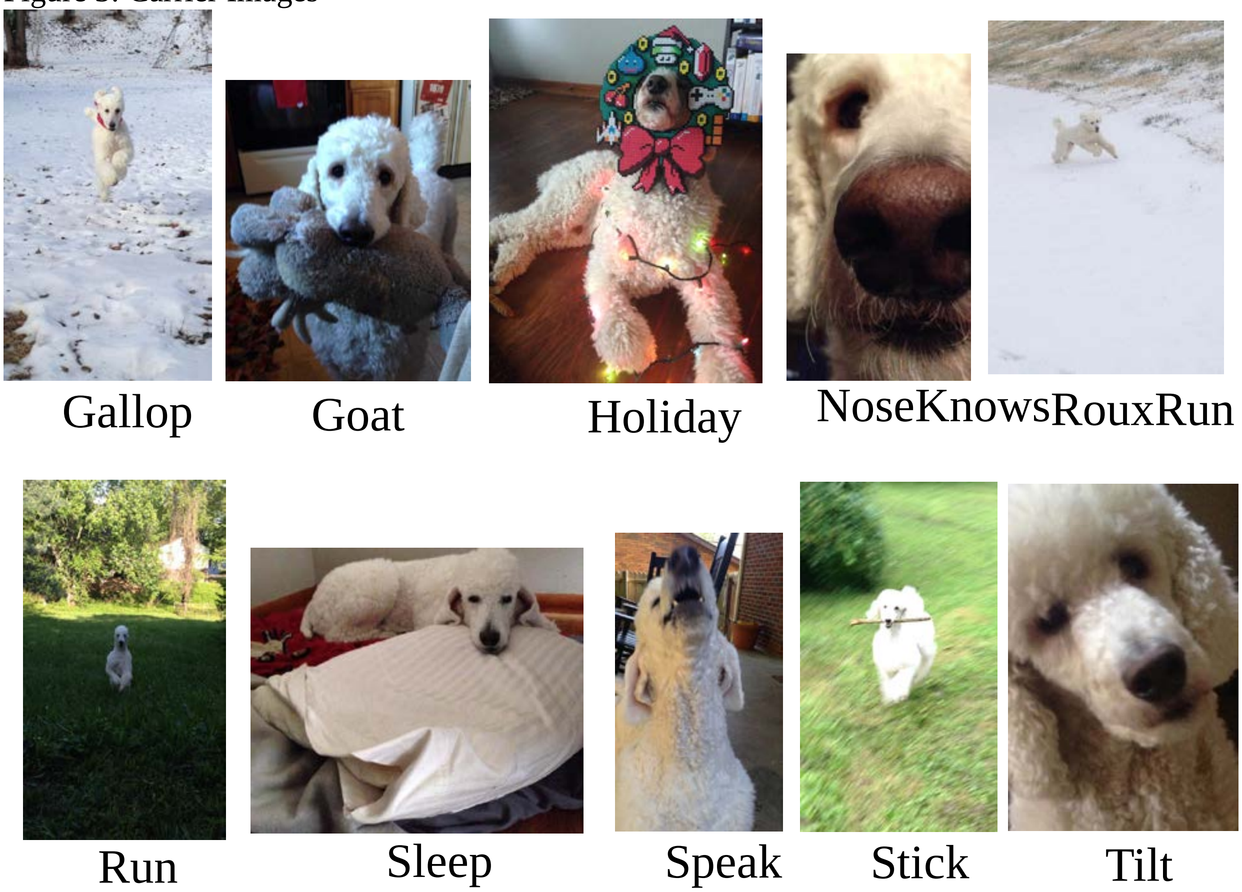
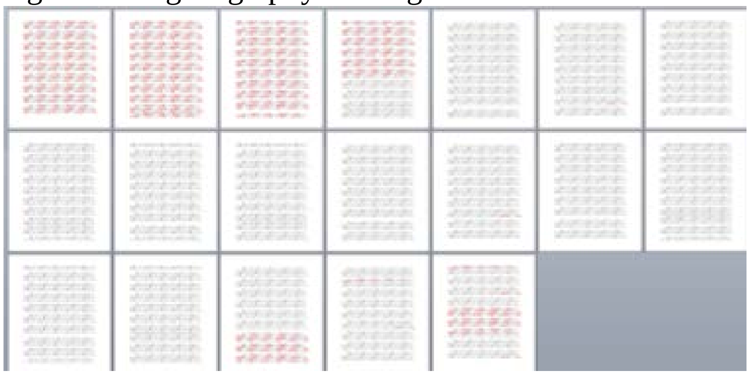
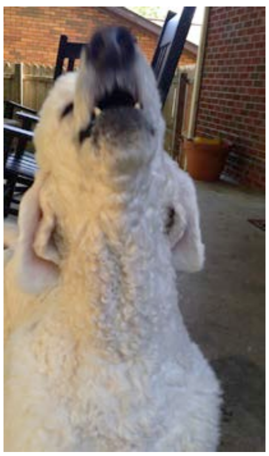


Figure 4: Steganography Message Files



RouxRunLge.doc

RouxRunSml.doc

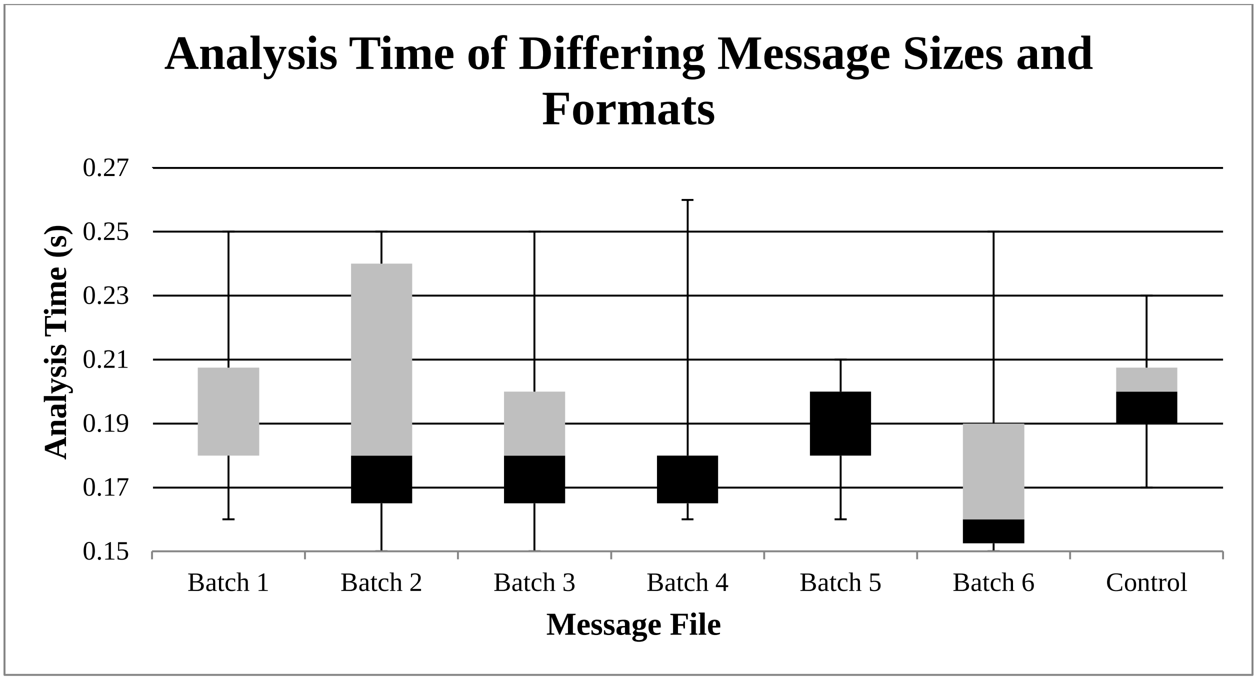


StegoLge.jpg, 9.764 MB
StegoSml.jpg, 1.007 MB

StegoLge.png, 9.740 MB
StegoSml.png, 1.003 MB

The steganography size specifications, image details, and raw results are provided in Appendices B and C. Figures 5 and 6 are the resulting relationships between the control and image analysis times, and are organized based on treatment group (all described in Appendices B and C). Figure 5 represents the trend-lines of each batch and corresponding control.

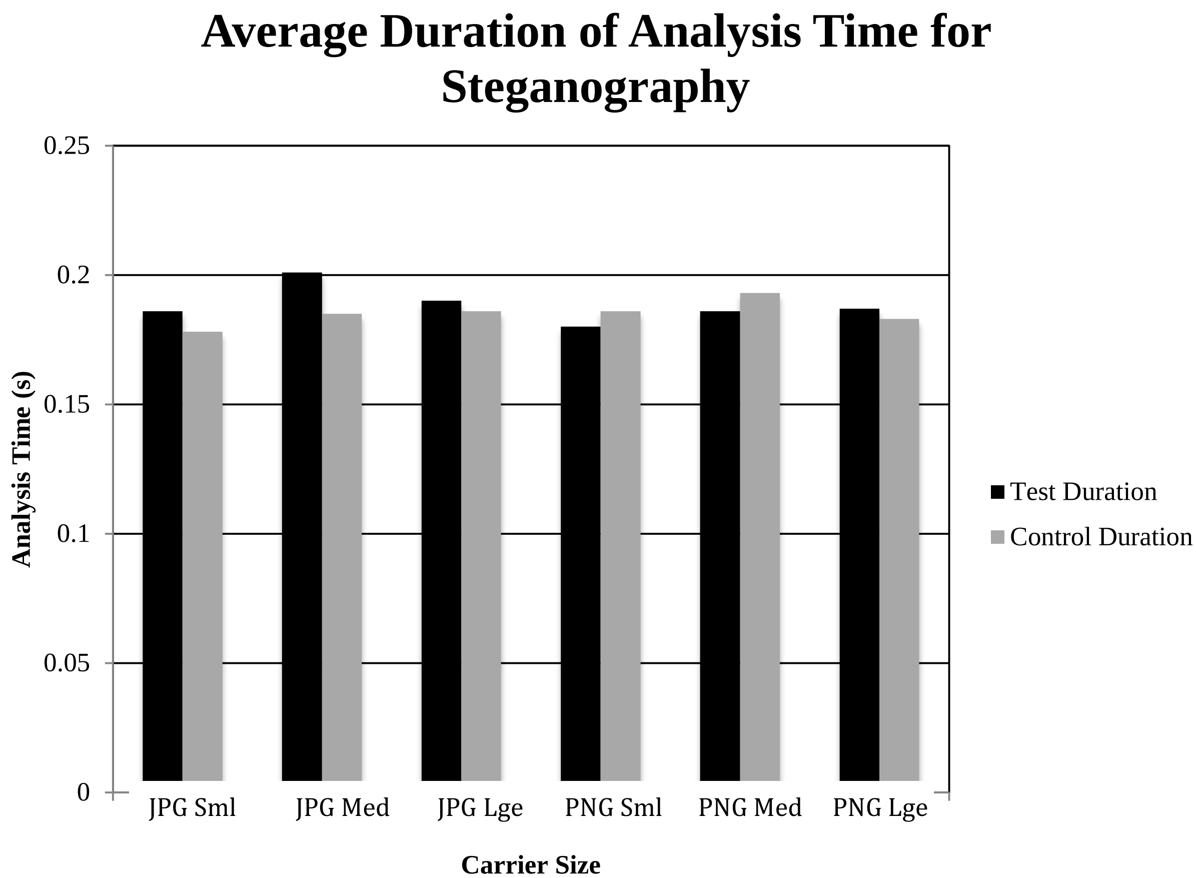
Figure 5: Method 1, StegAlyzer™ Analysis-Time of Steganography with Different Message Sizes and Formats



Run-times of analyzed images. Carrier images were 5 MB JPEGs (n = 10). Batch 1 was embedded with a 34 KB .doc file; Batch 2, a 103 KB .doc file; Batch 3, a 1 MB JPG file; Batch 4, a 10 MB JPG image; Batch 5, a 1 MB PNG image; Batch 6, a 10 MB PNG image. The control had no embedded media

Figure 6 is the average of the durations from each group compared to the average of the same size with no steganography embedded within. The largest steganography files were 14.7 MB. The smallest steganography file was 4.88MB. As shown in Appendix A, the size categories ranged from 0.98 to 1.03 (1 MB size), 4.85 to 4.92 (5 MB size), and 9.73 to 9.80 (10 MB size).

Figure 6: Method 2, Average StegAlyzerSS^{TM} Analysis-Time of Steganography with Different Carrier Sizes and Formats



The average run-times for each group of images (n = 10). Experimental images were embedded with the same JPG image, 5 MB in size. JPG Sml and PNG Sml represent an image size of 1 MB of corresponding image formats; JPG Med and PNG Med were 5 MB in size; JPG Lge and PNG Lge were images 10 MB in size. Controls had no embedded message images.

Figures 5 and 6 also provide detection durations for each image. Note that none of the analyses lasted longer than half of one second. The longest analysis was 0.26 of a second from Method 1 (RunMed.jpg with embedded StegoLge.jpg message) and the shortest was 0.15 of a second from multiple images from both Method 1 and 2.

With such a limited range of analysis duration, 0.11 of a second, statistical analysis has a likelihood of misrepresenting the impact of variation within analysis time. That is, in a practical setting the difference between 0.15 of a second and 0.26 of a second may go unnoticed, where a statistical analysis may suggest a significant difference between the two values depending on population size. That being said, statistical analyses of the data are provided in Table 1. The averages of the groups ranged from 0.174 of a second (Batch 6 from Method 1) to 0.201 of a second (Batch 2 from Method 2). The F value for Method 1 was 0.870643 for an F critical of 2.24641. The F value for Method 2 was 0.549979 for an F critical of 1.87838. This suggests that neither method differed significantly enough for the batches to be considered unique from each other. Qualitatively, each analysis seemed to be instantaneous.

Table 1: Statistical Analyses of StegalyzerSS™ Steganography Detection Duration

Groups	*Count*	*Sum*	*Average*	*Variance*
Method 1				
Batch 1	10	1.93	0.193	0.000734444
Batch 2	10	1.99	0.199	0.001565556
Batch 3	10	1.87	0.187	0.000867778
Batch 4	10	1.87	0.187	0.001067778
Batch 5	10	1.9	0.19	0.000266667
Batch 6	10	1.74	0.174	0.001071111
Control	10	1.99	0.199	0.000387778
Method 2				
JPGSml	10	1.86	0.186	0.000293333
JPGMed	10	2.01	0.201	0.000432222
JPGLge	10	1.9	0.19	0.000488889
PNGSml	10	1.8	0.18	0.000177778
PNGMed	10	1.86	0.186	0.000426667
PNGLge	10	1.87	0.187	0.000312222
CtrlJPGSml	10	1.87	0.187	0.000312222
CtrlJPGMed	10	1.92	0.192	0.000706667
CtrlJPGLge	10	1.88	0.188	0.000573333
CtrlPNGSml	10	1.9	0.19	0.000466667
CtrlPNGMed	10	1.92	0.192	0.000706667
CtrlPNGLge	10	1.92	0.192	0.000662222

ANOVA	df	F	P-Value	F-crit
Method 1				
Between Groups	6	0.870643	0.521549	2.24641
Within Groups	63			
Total	69			
Method 2				
Between Groups	11	0.549979	0.864675	1.87838
Within Groups	108			
Total	119			

Scenario 2:

Of the applications StegAlyzerAS™ scanned, five out of the nine applications were discovered: Steghide, SilentEye, OpenPuff, Virtual, GhostHost, and Steganography Studio. The full details of the applications including the method of obfuscation, carrier file format, and ability of StegAlyzerAS™ to discover it are detailed in Table 2.

Table 2: Application Analysis by StegAlyzerAS™

Application	Embed Method	Embed within	StegAlyzerAS™ Detection
SecretLayer	LSB	PNG	No
SilentEye	LSB	BMP, WAV	Yes
GhostHost	Append	All images, audio, text, and video	Yes
ImageSpyerG2	LSB - robust soliton distribution	BMP, TIF	No
OpenPuff	LSB - non-linear coding	BMP, JPG, PCX, PNG, TGA, AIFF, MP3, NEXT/SUN, WAV, 3GP, MP4, MPG, VOB, FLV, SWF, PDF	Yes
OpenStego	LSB, Watermarking	JPG, TXT, PNG, BMP	No
Steg	LSB	JPG, TIF, PNG, BMP, PPM	No

SteganographyStudio	LSB	BMP, PNG, GIF	Yes
Steghide	LSB – non-linear coding	JPG, BMP, WAV, AU	Yes

Scenario 3:

StegAlyzerSS™ detected two of the six steganography images (ImageSpyerG2.bmp and GhostHost.jpg). Steganography Studio crashed during each of three attempts to embed a message into the carrier, and so was removed from analysis. Details of the steganography sizes, carrier image and size, message file and size, and capability of identification via StegAlyzerSS™ are depicted in Table 3.

Table 3: Steganography Analysis by StegAlyzerSS™

Steganography Application	**Cover Image**	**Message**	**Steganography File**	**StegAlyzerSS™ Detection**
SecretLayer	StegoLge.jpg, 9.764 MB	RouxRunLge .doc, 105 KB	SecretLayer.jpg, 9.764 MB	No
GhostHost	StegoLge.jpg, 9.764 MB	RouxRunLge .doc, 105 KB	GhostHost.jpg, 9.867 MB	SS, Append
ImageSpyer G2	StegoLge.jpg, 9.764 MB	Roux.txt, 1 KB	ImageSpyerG2.bmp, 62.53 MB	LSB
OpenPuff	StegoLge.jpg, 9.764 MB	Roux.txt, 1 KB	OpenPuff.jpg, 9.764 MB	No
OpenStego	StegoLge.jpg, 9.764 MB	RouxRunLge .doc, 105 KB	OpenStego.png, 9.740 MB	No
Steg	StegoLge.jpg, 9.764 MB	RouxRunLge .doc, 105 KB	Steg.jpg, 9.762 MB	No
Steganography Studio	NA	NA	NA	NA

Conclusions

The concept of hiding information in plain sight is not new to civilization and the current, most efficient method of doing so is through steganography. Today, steganography has the potential of being a sophisticated and highly effective means of hiding information, licit or not. As steganography becomes more and more popular with criminals, data hiding becomes easier and more diverse. Various steganalysis programs seek to demystify this new method of information hiding, and in so doing thwart would-be criminals. StegAlyzer™ has the ability to discover steganography applications and to unveil their messages. Fortunately, investigations are not encumbered by the analysis of files that vary in size and complexity, as images as large as 10 MB are detected and analyzed in less than one quarter of a second. Further, StegAlyzerAS™ discovered the majority of applications searched in this investigation and steganography was uncovered 33% of the time by StegAlyzerSS™. In all, StegAlyzer™ proved to be an efficient program, though there is work to be done in regards to detecting the plethora of steganography applications on the Web, any of which may be installed on a given user's home computer. Additional research should investigate the comparison of StegAlyzer™ to various steganalysis tools available such as Wetstone's StegoHunt™ and Steganography Studio's [16] steganalysis function, the abilities of steganalysis tools to detect and decrypt non-linear RSD steganography, and StegAlyzer™'s detection capabilities with files much larger than those investigated in the current study.

Acknowledgements

We thank Backbone Security, specifically Chad Davis, for his support and correspondence throughout the investigation. We also thank Dr. Lauren Richards-Waugh for statistical assistance. Finally, we thank Roux, the poodle, for donating his images for use in this project.

Limitations of Study:

The StegAlyzer™ software was procured as a 30-day trial and thus may not have functioned completely akin to the full, licensed product.

References

[1] Aggarwal S, Jaiswal U. Kryptos+Graphein= Cryptography. Int J Eng Sci Technol 2011;3(9):7080-4.

[2] Ashok J, Raju Y, Munishankaraiah S, Srinivas K. Steganography: An Overview. Int J Eng Sci Technol 2010;2(10):5985-92.

[3] Atoum MS, Ibrahim S, Sulong G, M-Ahmad A. MP3 Steganography: Review. Int J Comput Sci 2012;9(6):236-44.

[4] Boora M, Ghambir F. Binary Image Steganography. Int J Recent Technol Eng 2013;2(5):126-31.

[5] Cheddad A, Condell J, Curran K, McKevitt P. Digital Image Steganography: Survey and analysis of current methods. Sign Proc 2010;90(3):728-50.

[6] Fogie S. Steganography. Informit.com. Pearson Education 2014. Accessed: June 25, 2014.

[7] Gadichal AB. Audio Wave Steganography. Int J Soft Comput Eng 2011;1(5):174-6.

[8] Image SpyerG2. ITNTSRL. http://imagespyer-g2.soft32.com/. Accessed June 03, 2014.

[9] Open Puff. Embedded SW. http://embeddedsw.net/OpenPuff_Steganography_Home.html. Accessed June 03, 2014.

[10] Open Stego. GNU. http://www.openstego.info/. Accessed June 03, 2014.

[11] Raphael J, Sundaram V. Cryptography and Steganography – A Survey. Int J Comput Tech Appl 2011;2(3):626-30.

[12] Secret Layer Steganography. Easy Sector. http://www.steganographypro.com/. Accessed June 03, 2014.

[13] Silent Eye. http://www.silenteye.org/. Accessed June 03, 2014.

[14] Steg. Drupal Gardens. http://steg.drupalgardens.com/. Accessed June 03, 2014.

[15] Steganography Analysis and Research Center. www.sarc-wv.com. BackBone Security 2014. Accessed: June 03, 2014.

[16] Steganography Studio. Source Forge. http://stegstudio.sourceforge.net/. Accessed June 03, 2014.

[17] Steghide. http://steghide.sourceforge.net/. Accessed June 03, 2014.

[18] Yugala K. Steganography. Int J Eng Trends Technol 2013;4(5):1629-35.

Appendix A: Parent Image and Message Sizes and Formats

File Name	Extension	Size (MB)	File Name	Extension	Size (MB)
GallopLge	.jpg	9.80	RunLge	.jpg	9.74
GallopMed	.jpg	4.87	RunMed	.jpg	4.86
GallopSml	.jpg	1.01	RunSm	.jpg	1.01
GoatLge	.jpg	9.76	SleepLge	.jpg	9.80
GoatMed	.jpg	4.85	SleepMed	.jpg	4.91
GoatSml	.jpg	0.98	SleepSm	.jpg	0.98
HolidayLge	.jpg	9.81	SpeakLge	.jpg	9.76
HolidayMed	.jpg	4.85	SpeakMed	.jpg	4.89
HolidaySml	.jpg	1.01	SpeakSm	.jpg	0.99
NoseKnowsLge	,jpg	9.76	StickLge	.jpg	9.78
NoseKnowsMed	.jpg	4.89	StickMed	.jpg	4.87
NoseKnowsSml	.jpg	1.01	StickSml	.jpg	1.02
RouxRunLge	.jpg	9.74	TiltLge	.jpg	9.80
RouxRunMed	.jpg	4.92	TiltMed	.jpg	4.90
RouxRunSml	.jpg	1.01	TiltSml	.jpg	0.99
GallopLge	.png	9.73	RunLge	.png	9.76
GallopMed	.png	4.86	RunMed	.png	4.92
GallopSml	.png	1.01	RunSm	.png	1.00
GoatLge	.png	9.77	SleepLge	.png	9.77
GoatMed	.png	4.90	SleepMed	.png	4.90
GoatSml	.png	1.01	SleepSm	.png	1.01
HolidayLge	.png	9.73	SpeakLge	.png	9.74
HolidayMed	.png	4.85	SpeakMed	.png	4.85
HolidaySml	.png	1.03	SpeakSm	.png	1.00
NoseKnowsLge	.png	9.76	StickLge	.png	9.77
NoseKnowsMed	.png	4.87	StickMed	.png	4.90
NoseKnowsSml	.png	1.00	StickSml	.png	1.01
RouxRunLge	.png	9.76	TiltLge	.png	9.80
RouxRunMed	.png	4.91	TiltMed	.png	4.85
RouxRunSml	.png	1.02	TiltSml	.png	1.02
RouxRunSm	.doc	0.03	StegoLge	.jpg	9.76
RouxRunLg	.doc	0.105	StegoSml	.png	1.00
StegoSml	.jpg	1.01	StegoLge	.png	9.74
StegoMed	.jpg	4.89			

Appendix B: Embedded Steganography Image Sizes and Formats of Method 1

Carrier Name	File Ext.	Message File	Stego Size (MB)	StegAlyzer™ Analysis Time (s)
Control				
GallopMed	JPG	None	None	0.19
GoatMed	JPG	None	None	0.20
HolidayMed	JPG	None	None	0.20
NoseKnowsMed	JPG	None	None	0.19

RouxRunMed	JPG	None	None	0.19
RunMed	JPG	None	None	0.17
SleepMed	JPG	None	None	0.23
SpeakMed	JPG	None	None	0.18
StickMed	JPG	None	None	0.21
TiltMed	JPG	None	None	0.23
Batch 1				
GallopMed	JPG	.doc, 34 KB	4.91	0.25
GoatMed	JPG	.doc, 34 KB	4.89	0.21
HolidayMed	JPG	.doc, 34 KB	4.88	0.21
NoseKnowsMed	JPG	.doc, 34 KB	4.92	0.16
RouxRunMed	JPG	.doc, 34 KB	4.96	0.16
RunMed	JPG	.doc, 34 KB	4.89	0.18
SleepMed	JPG	.doc, 34 KB	4.95	0.20
SpeakMed	JPG	.doc, 34 KB	4.92	0.18
StickMed	JPG	.doc, 34 KB	4.90	0.18
TiltMed	JPG	.doc, 34 KB	4.93	0.20
Batch 2				
GallopMed	JPG	.doc, 103 KB	4.98	0.25
GoatMed	JPG	.doc, 103 KB	4.95	0.16
HolidayMed	JPG	.doc, 103 KB	4.95	0.20
NoseKnowsMed	JPG	.doc, 103 KB	4.99	0.25
RouxRunMed	JPG	.doc, 103 KB	5.02	0.21
RunMed	JPG	.doc, 103 KB	4.96	0.15
SleepMed	JPG	.doc, 103 KB	5.02	0.25
SpeakMed	JPG	.doc, 103 KB	4.99	0.18
StickMed	JPG	.doc, 103 KB	4.97	0.18
TiltMed	JPG	.doc, 103 KB	5.00	0.16
Batch 3				
GallopMed	JPG	.jpg, 1.01MB	5.88	0.18
GoatMed	JPG	.jpg, 1.01 MB	5.86	0.18
HolidayMed	JPG	.jpg, 1.01 MB	5.86	0.21
NoseKnowsMed	JPG	.jpg, 1.01 MB	5.89	0.25
RouxRunMed	JPG	.jpg, 1.01 MB	5.93	0.18
RunMed	JPG	.jpg, 1.01 MB	5.86	0.20
SleepMed	JPG	.jpg, 1.01 MB	5.92	0.16
SpeakMed	JPG	.jpg, 1.01 MB	5.90	0.20
StickMed	JPG	.jpg, 1.01 MB	5.87	0.16
TiltMed	JPG	.jpg, 1.01 MB	5.90	0.15
Batch 4				
GallopMed	JPG	.jpg, 9.76 MB	14.6	0.18
GoatMed	JPG	.jpg, 9.76 MB	14.6	0.18
HolidayMed	JPG	.jpg, 9.76 MB	14.6	0.16
NoseKnowsMed	JPG	.jpg, 9.76 MB	14.7	0.16
RouxRunMed	JPG	.jpg, 9.76 MB	14.7	0.18
RunMed	JPG	.jpg, 9.76 MB	14.6	0.26
SleepMed	JPG	.jpg, 9.76 MB	14.7	0.18

SpeakMed	JPG	.jpg, 9.76 MB	14.7	0.16
StickMed	JPG	.jpg, 9.76 MB	14.6	0.23
TiltMed	JPG	.jpg, 9.76 MB	14.7	0.18
Batch 5				
GallopMed	JPG	.png, 1.00 MB	5.76	0.18
GoatMed	JPG	.png, 1.00 MB	5.85	0.20
HolidayMed	JPG	.png, 1.00 MB	5.85	0.18
NoseKnowsMed	JPG	.png, 1.00 MB	5.90	0.20
RouxRunMed	JPG	.png, 1.00 MB	5.92	0.16
RunMed	JPG	.png, 1.00 MB	5.86	0.21
SleepMed	JPG	.png, 1.00 MB	5.92	0.20
SpeakMed	JPG	.png, 1.00 MB	5.89	0.18
StickMed	JPG	.png, 1.00 MB	5.87	0.21
TiltMed	JPG	.png, 1.00 MB	5.90	0.18
Batch 6				
GallopMed	JPG	.png, 9.74 MB	14.6	0.25
GoatMed	JPG	.png, 9.74 MB	14.6	0.16
HolidayMed	JPG	.png, 9.74 MB	14.6	0.15
NoseKnowsMed	JPG	.png, 9.74 MB	14.6	0.16
RouxRunMed	JPG	.png, 9.74 MB	14.7	0.20
RunMed	JPG	.png, 9.74 MB	14.6	0.15
SleepMed	JPG	.png, 9.74 MB	14.7	0.20
SpeakMed	JPG	.png, 9.74 MB	14.6	0.15
StickMed	JPG	.png, 9.74 MB	14.6	0.16
TiltMed	JPG	.png, 9.74 MB	14.6	0.16

Appendix C: Steganography Image Size, Format, and Analysis Time with Method 2

Steganography Image	Size (MB)	Analysis Time (s)	Steganography Image	Size (MB)	Analysis Time (s)
JPG			**PNG**		
Control			Control		
GallopSml	5.90	0.20	GallopSml	5.90	0.15
GoatSml	5.90	0.18	GoatSml	5.90	0.18
HolidaySml	5.90	0.20	HolidaySml	5.91	0.21
NoseKnowsSml	5.90	0.16	NoseKnowsSml	5.89	0.18
RouxRunSml	5.90	0.20	RouxRunSml	5.91	0.18
RunSml	5.90	0.18	RunSml	5.88	0.18
SleepSml	5.87	0.21	SleepSml	5.90	0.23
SpeakSml	5.88	0.20	SpeakSml	5.89	0.19
StickSml	5.91	0.18	StickSml	5.90	0.20
TiltSml	5.88	0.16	TiltSml	5.91	0.20
Control			Control		
GallopMed	9.76	0.16	GallopMed	9.75	0.16
GoatMed	9.74	0.18	GoatMed	9.78	0.21
HolidayMed	9.74	0.16	HolidayMed	9.74	0.16

NoseKnowsMed	9.78	0.25	NoseKnowsMed	9.75	0.25
RouxRunMed	9.81	0.20	RouxRunMed	9.80	0.18
RunMed	9.75	0.21	RunMed	9.80	0.20
SleepMed	9.80	0.18	SleepMed	9.80	0.20
SpeakMed	9.78	0.18	SpeakMed	9.74	0.20
StickMed	9.76	0.20	StickMed	9.79	0.18
TiltMed	9.79	0.20	TiltMed	9.74	0.18
Control			Control		
GallopLge	14.7	0.16	GallopLge	14.6	0.20
GoatLge	14.7	0.18	GoatLge	14.7	0.20
HolidayLge	14.7	0.20	HolidayLge	14.6	0.18
NoseKnowsLge	14.7	0.16	NoseKnowsLge	14.7	0.16
RouxRunLge	14.6	0.20	RouxRunLge	14.6	0.18
RunLge	14.7	0.21	RunLge	14.7	0.20
SleepLge	14.7	0.18	SleepLge	14.7	0.20
SpeakLge	14.7	0.20	SpeakLge	14.6	0.19
StickLge	14.7	0.23	StickLge	14.7	0.25
TiltLge	14.7	0.16	TiltLge	14.7	0.16
JPG			**PNG**		
Batch 1			Batch 4		
GallopSml	5.90	0.20	GallopSml	5.90	0.18
GoatSml	5.90	0.15	GoatSml	5.90	0.20
HolidaySml	5.90	0.18	HolidaySml	5.91	0.16
NoseKnowsSml	5.90	0.18	NoseKnowsSml	5.89	0.18
RouxRunSml	5.90	0.18	RouxRunSml	5.91	0.20
RunSml	5.90	0.20	RunSml	5.88	0.18
SleepSml	5.87	0.20	SleepSml	5.90	0.16
SpeakSml	5.88	0.18	SpeakSml	5.89	0.18
StickSml	5.91	0.18	StickSml	5.90	0.18
TiltSml	5.88	0.21	TiltSml	5.91	0.18
Batch 2			Batch 5		
GallopMed	9.76	0.20	GallopMed	9.75	0.16
GoatMed	9.74	0.25	GoatMed	9.78	0.21
HolidayMed	9.74	0.18	HolidayMed	9.74	0.16
NoseKnowsMed	9.78	0.18	NoseKnowsMed	9.75	0.18
RouxRunMed	9.81	0.20	RouxRunMed	9.80	0.16
RunMed	9.75	0.21	RunMed	9.80	0.20
SleepMed	9.80	0.20	SleepMed	9.80	0.20
SpeakMed	9.78	0.20	SpeakMed	9.74	0.21
StickMed	9.76	0.21	StickMed	9.79	0.18
TiltMed	9.79	0.18	TiltMed	9.74	0.20
Batch 3			Batch 6		
GallopLge	14.7	0.18	GallopLge	14.6	0.21
GoatLge	14.7	0.16	GoatLge	14.7	0.16
HolidayLge	14.7	0.23	HolidayLge	14.6	0.16
NoseKnowsLge	14.7	0.21	NoseKnowsLge	14.7	0.20

RouxRunLge	14.6	0.20	RouxRunLge	14.6	0.18
RunLge	14.7	0.20	RunLge	14.7	0.18
SleepLge	14.7	0.16	SleepLge	14.7	0.20
SpeakLge	14.7	0.18	SpeakLge	14.6	0.18
StickLge	14.7	0.18	StickLge	14.7	0.20
TiltLge	14.7	0.20	TiltLge	14.7	0.20